MaPPeRTrac: A Massively Parallel, Portable, and Reproducible Tractography Pipeline

A collaboration between the U.S. Department of Energy and TRACK-TBI[1]

Joseph Moon[a], Peer-Timo Bremer[b], Pratik Mukherjee[b], Amy J. Markowitz[b], Eva M. Palacios[b], Alexis Rodriguez[c], Yuaki Xiao[c], Geoffrey T. Manley[b], Ravi K. Madduri[c]

[a] *Lawrence Livermore National Laboratory*
[b] *University of California, San Francisco*
[c] *Argonne National Laboratory*

Abstract

Large scale diffusion MRI tractography remains a significant challenge. Users must orchestrate a complex sequence of instructions that require many software packages with complex dependencies and high computational cost. We developed MaPPeRTrac, a diffusion MRI tractography pipeline that simplifies and vastly accelerates this process on a wide range of high performance computing environments. It fully automates the entire tractography workflow, from management of raw MRI machine data to edge-density visualization of the connectome. Data and dependencies, handled by the *Brain Imaging Data Structure* (BIDS) and *Containerization using Docker and Singularity*, are de-coupled from code to enable rapid prototyping and modification. Data artifacts are designed to be findable, accessible, interoperable, and reusable in accordance with FAIR principles. The pipeline takes full advantage of HPC resources using the *Parsl* parallel programming framework, resulting in creation of connectome datasets of unprecedented size. MaPPeRTrac is publicly available and tested on commercial and scientific hardware, so that it may accelerate brain connectome research for a broader user community.

Keywords: Connectomes, Image Processing, High Performance Computing, FAIR

[1] Transforming Research and Clinical Knowledge in Traumatic Brain Injury

1. Introduction

Diffusion MRI can be digitally processed into a tensor field describing white matter fiber orientation *in-vivo*. By Monte Carlo sampling on this field, it is possible to estimate axon pathways between regions of the brain [1]. The resulting graph, known as a structural connectome, provides a quantitative measure of brain connectivity useful to analyze both healthy brains and the changes of neurological and psychiatric diseases. Connectomes can be evaluated using a plethora of well-developed techniques based on graph theory and matrix analysis. As a result, they show great promise towards the timely diagnosis and treatment of disorders of white matter connectivity such as traumatic brain injury (TBI) [2].

Recent advances in diffusion MRI have enabled the creation of vast high-resolution datasets [3]. However, connectome research has been limited by the throughput of neuroimaging software. Existing tools tend to need to resolve and install complicated prerequisite tools that are difficult to install and run at new computing sites, or do not support parallel and efficient execution. Generating connectomes for even a handful of patients can take days on a high-end personal computer and requires the dedicated attention of an operator skilled in neuroimaging and computer science [4]. As a result, dealing with cohorts of hundreds or thousands of patients is virtually impossible for most research groups. Addressing this challenge requires a neuroimaging pipeline that can easily be deployed, designed to use high performance computing (HPC) for efficient execution and leverage existing standards that can ensure reproducibility. When designing a tractography pipeline, several competing needs must be considered.

1.1. Need for high performance

Analysis methods should be designed and implemented using appropriate software constructs to take advantage of high performance computational resources. This is especially the case for pipeline instructions that have a memory footprint larger than what the standard computers can handle. The Department of Energy is home to some of the world's fastest supercomputers, opening a novel opportunity for large-scale tractography [5]. Parallelization and high performance go hand in hand, so understanding the inherent parallelization in the analysis will lead to vastly increased efficiency. One must pay close attention to potential parallelism at every step in the tractography pipeline, so that it may reduce the steep computational cost.

1.2. Need for reproducibility

Regardless of performance, any tractography workflow must produce results that are consistent across multiple execution environments and accurately represent the intentions of the analysts. For probabilistic tractography, this can be divided into processes that are deterministic and processes that are stochastic in nature. Deterministic algorithms, such as brain extraction and segmentation, should produce identical outputs given same inputs on all computing environments. Any stochastic process - in this case, the tractography itself - must produce results that are roughly similar given identical inputs, with values converging as the number of samples increases. Reproducibility is particularly important for connectome analysis, since it remains difficult to validate the findings of graph theoretical methods merely with neurological observation [6].

1.3. Need for portability

We define portability as the ability to run on any available computational resources and computational environments. Portability is key for reusability as well as wider adoption and deployment. All of the high performance optimizations should not be limited to supercomputing resources, but translate well to non-supercomputing platforms. It is also important the input and output data be easily shared between users. As far as possible, data should be organized in standard, machine-readable structures and use consistent file formats. Similar adoption of best practices, community standards should also be used for metadata and naming conventions, so that individual files can be more easily identified and manipulated. Lastly, the design and implementation of the pipeline should follow best practices of object oriented programming techniques and enable customization by configuration files so the pipeline can easily be extended and modified to suit different users and sites.

There exist several neuroimaging pipelines for diffusion MRI, but only a few take advantage of HPC capabilities. This is especially true at the largest scale, where computation is coupled to sophisticated resource allocation algorithms and distributed across multiple different clusters. The LONI Processing Environment provides an interactive neuroimaging workspace with fully modular components [7]. However, it has several limitations: HPC execution is only possible with the Grid Scheduler, heterogeneous resources such as GPUs are poorly supported, and software dependencies must be installed manually with elevated-user privileges. Schirner et al. [8] propose a multimodal pipeline that can generate connectomes on HPC clusters. Though

closely related, it does not meet all of our requirements. The pipeline's imperative software architecture makes data and scripts brittle and difficult to modify. There is no support for GPU acceleration or low-level parallelism, with the result that processing even a single subject exceeds time limitations on certain HPC clusters. And similar to the LONI pipeline, it demands root privileges to install software libraries. There exist non-modular diffusion MRI pipelines that *do* meet most of our platform requirements. However, they use deterministic instead of probabilistic tractography or process functional rather than structural connectomes, and the tight coupling of pipeline code makes them impractical to adapt to our parameters [9, 10].

1.4. Summary of MaPPeRTrac contributions

Our objective is to perform tractography analysis - from raw signal processing to generating the final connectome - while taking full advantage of HPC resources. Code and dependent libraries ought to be portable across computing systems, including those where root privileges are not available. Neuroimaging parameters and resource allocation should be configurable and consistent across different software components. Deterministic algorithms must be reproducible across systems. And most significantly, the pipeline should be as performant as possible in a parallel computing environment. To this end, we developed MaPPeRTrac, a **m**assively **p**arallel, **p**ortable, and **r**eproducible tractography pipeline that is capable of:

1. Leveraging parallelization: Exploit the inherent parallelism in neuroimaging analysis, not only across participants but also within individual analyses;
2. Portability across computing infrastructures: Transparently execute neuroimaging analysis on computing infrastructures at different institutions, with minimal changes to the pipeline itself.
3. Reproducing deterministic results: Ensure that results at different computing sites are exactly equivalent (or closely comparable, for stochastic procedures such as tractography).
4. Configuring all neuroimaging parameters: Allow easy access to the various parameters used by neuroimaging components in the pipeline; and
5. Being FAIR: Ensure that datasets, software, and other digital objects are **F**indable, **A**ccessible, **I**nteroperable, and **R**eusable according to standard FAIR guidelines [11]. This facilitates long-term use in research, promotes knowledge integration, and increases reusability of existing data.

2. Materials and Methods

2.1. Subjects

The workflow has been tested extensively on a large cohort of real-world subjects. All participants were enrolled at eighteen Level 1 Trauma Centers across the USA as part of the prospective Transforming Research and Clinical Knowledge in Traumatic Brain Injury project (TRACK-TBI) [3]. TRACK-TBI is a National Institutes of Health–funded multi-center study that began in October 2013. The objective is to create a large, high-quality database that integrates standardized clinical, imaging, proteomic, genomic, and outcome measures to establish more precise methods for TBI diagnosis and prognosis. A total sample of 842 subjects were included in the study, 773 TBI patients and 109 orthopedic trauma controls without head injury. Patients were recruited and included within 24 hours after injury upon meeting the American Congress of Rehabilitation Medicine (ACRM) criteria for TBI. Other inclusion criteria were having a brain CT as part of clinical care within 24 hours of injury, no significant polytrauma that would interfere with their assessment, and no MRI contraindication. Exclusion included prior major psychiatric or neurological pathology.

Traumatic brain injury patients were recruited for the pilot phase of the TRACK-TBI study at three centers, n=150 (age: median =, SD±: education:median=years, SD±; sex F/M) (Yuh et al., 2014) and the rest of the patients for the current TRACK-TBI study at eleven centers, n=584 (age=, SD±: education: median=years, SD±; sex F/M). The TRACK-TBI orthopedic trauma control subjects, n=109 (age: $\bar{x}$=, SD±: education :years, SD±; sex F/M) were, like the patients, recruited from the Level 1 Trauma Centers for the TRACK-TBI study, and presented mainly with lower extremity fractures. Orthopedic controls were ruled out for the current study if the emergency room physician required CT scan for suspicion of head trauma, or, if by interviewing medical services or subjects, participants reported clinical information such as loss of consciousness, amnesia, previous TBI, or major psychiatric or neurological pathology. All eligible subjects who voluntarily agreed to participate gave written informed consent. All study protocols were approved by the Institutional Review Boards of the enrollment centers.

2.2. MRI Data Acquisitions

For the TRACK-TBI pilot phase study, the 3 Tesla (3T) MRI datasets performed 1-2 weeks after injury included whole-brain diffusion tensor imaging (DTI) that was performed with a multi-slice single-shot spin echo echo-

planar pulse sequence using 55 diffusion-encoding directions, isotropically distributed over the surface of a sphere with electrostatic repulsion, acquired at b=1000sec/mm2, seven acquisitions at b=0sec/mm2, 72 interleaved slices of 1.8-mm thickness each with no gap between slices, a 128x128 matrix, and a field of view (FOV) of 230x230mm. Additionally, conventional 3T MRI axial three-dimensional (3D) inversion recovery gradient recalled echo T1-weighted images were acquired with 230-mm FOV, 156 contiguous partitions (1.0-mm) at a 256x256 matrix. The current phase TRACK-TBI patients and orthopedic control subjects were scanned on 3T MRI scanners at 2 weeks ($\bar{x}$=13.30 days; SD±2.10) and 6 months ($\bar{x}$=184 days; SD±8.86) after injury. Whole-brain DTI was performed with a multi-slice single-shot spin echo echoplanar pulse sequence using 64 diffusion-encoding directions, isotropically distributed over the surface of a sphere with electrostatic repulsion, acquired at b = 1300 sec/mm2, eight acquisitions at b = 0 s/mm2, slices of 2.7-mm thickness each with no gap between slices, a 128x128 matrix, and a field of view (FOV) of 350x350 mm, resulting in 2.7-mm isotropic voxels. Sagittal 3D inversion recovery gradient recalled echo T1-weighted images were acquired with 256mm FOV, 200 contiguous partitions (1.2 mm) at a 256x256 matrix. All scans were visually inspected before any pre-processing or analysis. Thirty-six of the TBI patients and 2 orthopedic controls were excluded for having incomplete datasets.

3. Original Tractography Scripts

Our pipeline emerged from an existing collection of neuroimaging scripts by Owen et al. [12]. These scripts produce two outputs: 1) the grey matter connectivity adjacency matrix typical of a structural connectome and 2) edge density imaging (EDI), which maps the number of network edges that pass through every white matter voxel in the brain. These are generated through the following steps:

1. Read diffusion MRI data. Correct for motion and scan artifacts, remove non-brain tissue, and calculate diffusion anisotropy.
2. Estimate diffusion parameters and fiber directions in each voxel
3. Parcellate the cortical and subcortical gray matter into 82 regions based on the Desikan–Killiany atlas
4. Compute white matter fiber streamlines to generate a structural connectome using probabilistic tractography. Collate tractography into an EDI map.

The original scripts come with several limiting characteristics. Exact versions of software libraries - FSL, Freesurfer, and CUDA - must be installed locally. Input and output data must be stored at hard-coded paths. The user must specify resource allocation and submit batch jobs to the HPC scheduler manually. And parameters can only be adjusted by modifying the scripts themselves. Although the scripts are carefully tailored to run on specific HPC clusters, they are not sufficiently robust for anything more than experimentation with a handful of patients.

4. MaPPeRTrac

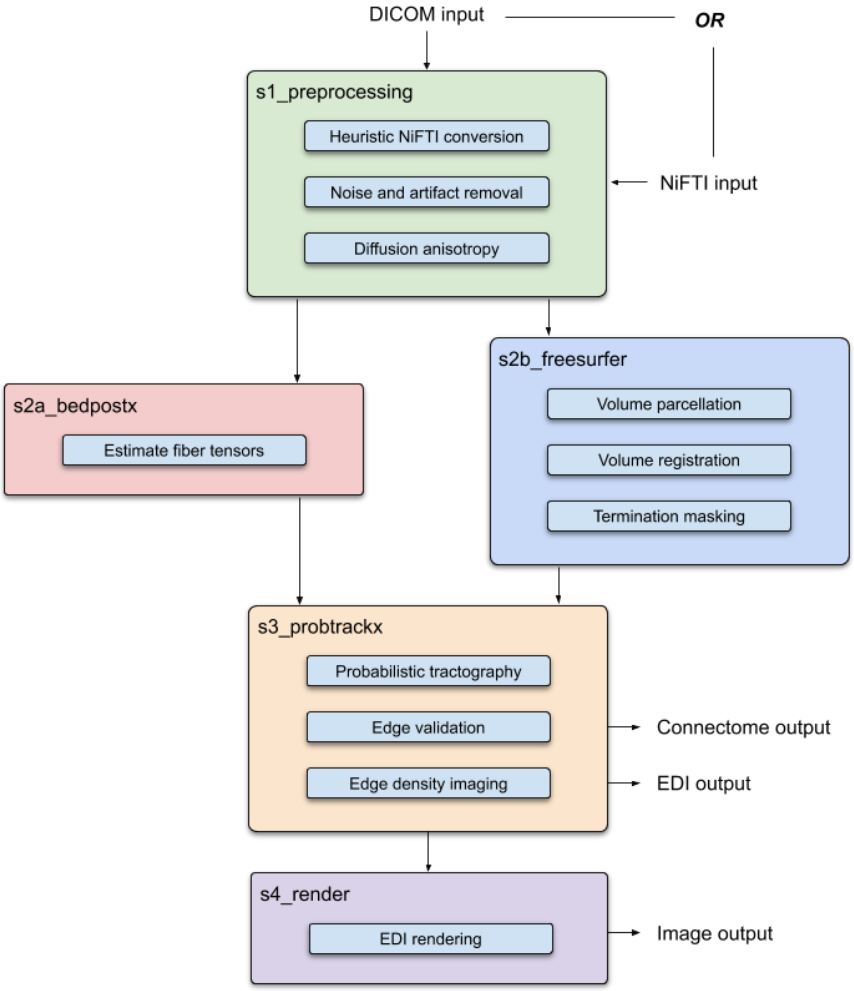

Figure 1: Overview of MaPPeRTrac architecture

MaPPeRTrac accomplishes the goals of the original tractography scripts, but with a significantly faster, more portable, and better parametrized implementation. It takes advantage of parallelism at both the subject-level and task-level coupled with GPU acceleration that is configurable, to significantly speed up connectome generation on HPC clusters. It also preprocesses raw DICOM files from a wide variety of MRI scanners, dynamically handles resource allocation and parameter configuration, and performs

heuristic validation on intermediate data outputs. This level of complexity and robust execution is managed by a clear functional hierarchy and containerization of software dependencies, ensuring that users can easily modify and deploy the tools on any HPC cluster.

4.1. MRI Data Preparation

DICOM scanner images were converted to NIfTI neuroimaging file format using *dcm2niix*. Given the wide variety of scanner models and software versions each of the three major MR manufacturers, specifically GE, Philips and Siemens, this step of the neuroimaging pipeline was quite laborious. We automated this step and integrated it into the pipeline, so that our pipeline accepts either DICOM or NIfTI diffusion MRI files from all vendors.

4.2. Pre-processing

One of the limitations of the original scripts is their inability to read data directly from MRI scanners. Converting raw DICOM files into a diffusion tensor image (DTI) and T1-weighted anatomical volume is a laborious process, even with automation tools. This is made more challenging by the wide variety of MRI scanners and file standards. MaPPeRTrac overcomes this obstacle by proposing a novel algorithm to universally pre-process DICOM files. It converts all DICOM files into NIfTI format and applies a series of statistical heuristics to determine whether to process them into the DTI, anatomical volume, or b-value weighting. As a result, our pipeline can handle inputs regardless of naming convention or file structure.

At times, pre-processing MRI inputs without human oversight runs the risk of faulty behavior, additional heuristics in the pipeline help catch invalid data. The pipeline alerts users if data are clearly missing or significant outliers are found. After manually reviewing 200 diffusion scans and connectomes generated by this pipeline, we find none that are handled incorrectly by this pre-processing step.

4.3. Parallelization

The original scripts for generating structural connectomes are a monolithic sequence of commands. We modularized the components of the pipeline and implemented parallel execution of the modules. In our efforts to modularize and parallelize the different analysis steps, we leveraged a parallel scripting language called Parsl[13] that is being developed at University of Chicago and is designed to process massive batches of data on heterogeneous computing resources. Parsl is especially designed to work seamlessly on various DOE computational resources. The modularization of various

analysis steps enabled us to perform profiling of various modules where we measured the computational(CPU, memory, I/O) requirements of individual steps, insights into parallelization possible by virtue of tracking data dependencies among different stages of the pipeline and helped define and tailor resource requirements for specific steps. Previous neuroimaging pipelines tend to reserve maximum computational resources, that remains unchanged throughout the execution of the pipeline, so that the most expensive steps can complete. Since this strategy may be wasteful or unavailable on shared computational resources, our pipeline requests a dynamic amount of resources appropriate to its tasks at time of execution.

Parsl defines the pipeline architecture using the data dependencies among the steps and dispatches tasks to the underlying schedulers tasks that have all the input datasets available. This feature enabled to exploit parallelism at pipeline level and at the individual task level. For example, if there are 1000 MRI input datasets available to process, all of the samples can be submitted at once and the pipeline would request resources to process the first step of the pipeline for all the samples in parallel. Subsequent steps of the pipeline are scheduled appropriately when the inputs of the steps are generated.In addition to pipeline-level parallelization, we implemented task level parallelization where tasks in the pipeline that do not have data dependencies get scheduled and executed in parallel. In MaPPeRTrac, this is manifested in parallel execution of steps that perform parcellation and estimation of diffusion tensor fields. The pipeline also takes advantage of sub-task level parallelization where we exploited the multiple CPU cores present in a compute node and performed the computation in parallel. This is achieved by dividing the input data into smaller chunks and running the analysis on individual CPU cores. The sub-task level paralleliztion is implemented in freesurfer based recon-all step using openMP (Reference) and in the probabilistic tractography step. Using sub-task level parallelization helped speedup the freesurfer step by a few hours. On an average the freesurfer /emphrecon-all step was taking 14-20 hours to complete and using sub-task level parallelization the execution time came to an average of 7-10hours.

4.4. Parametrization

The implementation of the pipeline using Parsl involved creation of a directed acyclic graph where the edges are data dependencies among the steps. Since all the steps in the pipeline are run without user intervention, the execution engine needs to have all the parameters for all the steps be available. The need for parametrization of individual steps is even greater

when executing a large batch of the MRI datasets as it is impractical to have a human operator check the outputs of a step and launch the subsequent step manually. Since the graph of tasks is specified when launching Parsl, this architecture necessitates parameterization of the elements required to describe the tasks and sub-tasks. All the pipeline parameters are configurable from a single source, which can either be command-line arguments or a configuration file. When left unspecified, the pipeline will estimate appropriate parameters or use default values from the original script collection. This enables users to easily orchestrate complex jobs across different HPC clusters.

4.5. Portability

The Parsl framework is implemented in the Python language which is one of the most popular cross platform programming language. The pipeline is implemented as a Python package so it can be used on any platform that supports Python. One of the strengths of the Parsl framework is the separation between definition of tasks and the actual execution of the tasks. Once the tasks are defined in Parsl, its plug-in based architecture enables execution on most of the modern HPC schedulers.

Parsl supports execution on computational clouds, supercomputers and HPC clusters seamlessly using plugins for various computational elements. Each provider implementation may allow users to specify additional parameters for further configuration. Parameters are generally mapped to resource manager submission script or cloud API options. Examples of local resource-manager-specific options are partition, wall clock time, scheduler options (e.g., #SBATCH arguments for Slurm), and worker initialization commands (e.g., loading a conda environment). Cloud parameters include access keys, instance type, and spot bid price. At the time of writing this manuscript, Parsl supports 12 *providers* that include the gamut of frameworks that deliver computational resources. To overcome the differences in these compute elements, and present a single uniform interface, Parsl implements a simple provider abstraction. This abstraction is key to Parsl's ability to enable scripts to be moved between resources. The provider interface exposes three core actions: submit a job for execution (e.g., sbatch for the Slurm resource manager), retrieve the status of an allocation (e.g., squeue), and cancel a running job (e.g., scancel). Parsl implements providers for local execution (fork), for various cloud platforms using cloud-specific APIs, and for clusters and supercomputers that use a Local Resource Manager (LRM) to manage access to resources, such as Slurm, HTCondor, and Cobalt. By leveraging

Parsl to define our pipeline, we have been able to achieve portability of execution.

After acheiving execution portability using Parsl, we have examined different ways in which we can create portable bundles of applications that can be packaged and used in conjunction of the pipeline. Examining past work in reproducible neuroscience such as Theaud et al. [9], we concluded that containerization technologies are key to ensure portability. Software containers are portable execution elements that provide an abstraction over underlying operating system and hardware to to improve the portability of applications. Containerization technologies are supported by computational clouds, supercomputers and campus HPC providers. Two popular containerization technologies exist with important technical differences. In order to support extensive portability, we generated containers using both Singularity and Docker technologies and made these containers publicly available. In addition to increasing the portability of the pipelines, the containerization technologies also enable versioning of various analysis tools that make up the pipeline. New versions of the analysis tools, for example, a new version of freesurfer, can easily be tested by updating the 'recipe' for generation of the container to point to the location of the updated version of freesurfer. Our recipes for Docker and Singularity containers contain all the tools and their dependencies for successful execution of the analysis. Users can either download an existing pre-built container or use the recipes to build it themselves. This is particularly important since a major hurdle to scientific software development is the onerous installation of tools and their dependencies.

The addition of numerous features and performance improvements to the pipeline reflects commensurate growth in the complexity of its software and output data. To mitigate complexity, we strictly adhere to the Brain Imaging Data Structure (BIDS), a protocol for standardizing neuroimaging data. The DICOM and NIfTI inputs are organized and shared along with tractography outputs, ensuring datasets across the entire pipeline remain transparent. Adoption of the BIDS data format addresses another key portability challenge of how the inputs are described and where the outputs of various steps of the pipeline are available.

5. Performance enhancements and other improvements

In addition to the performance improvements resulting from pipeline-level, task-level, sub-task level parallelization of the pipeline, we have made significant improvements in specific tools.

1. Dynamic memory management of Probtrackx: Probtrackx is very memory-intensive. Though Parsl has a wide range of features to manage resource allocation, the configuration and optimization of individual tasks has to be done using the computational profiling of the tool using different inputs. This has been challenging as the requirements varied across different datasets. Because we generate the inputs to Probtrackx at run-time, it is difficult to predict just how much memory each task will need. Overestimating memory usage prevents parallelization entirely, because the worst-case tasks can consume an entire node's memory. In order to run Probtrackx safely in parallel, we estimate each task's memory usage at run-time and record the total estimate for each node in a thread-safe file. New tasks attempting to launch Probtrackx will wait until their node's estimated memory usage falls below a safe threshold, found in testing to be between 50 to 70 percent of the node's total memory. After completing a Probtrackx run, each task updates the record to free up memory and allow other tasks to proceed.
2. Performance tracking code: We enhanced the pipeline by computational profiling code that measures core-time and wall-time of each step for each subject. The pipeline saves performance data and logs to the file system after each task is completed. Since each parallel process records data independently, we were able to collate performance data into a global timing log. This log, which contains resource usage and wall-clock time for each processing step, enables users to make informed estimates in future computations.
3. Automatic visualization: we wrote an additional pipeline step, to render edge density images generated by the pipeline. Image rendering is even possible on headless nodes as we included the VTK library with the container.

6. Results and Evaluation

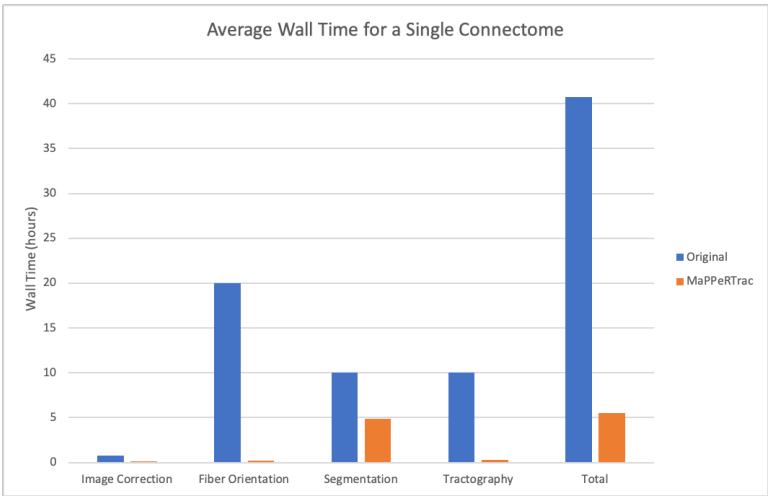

Figure 2: Average time for a single connectome

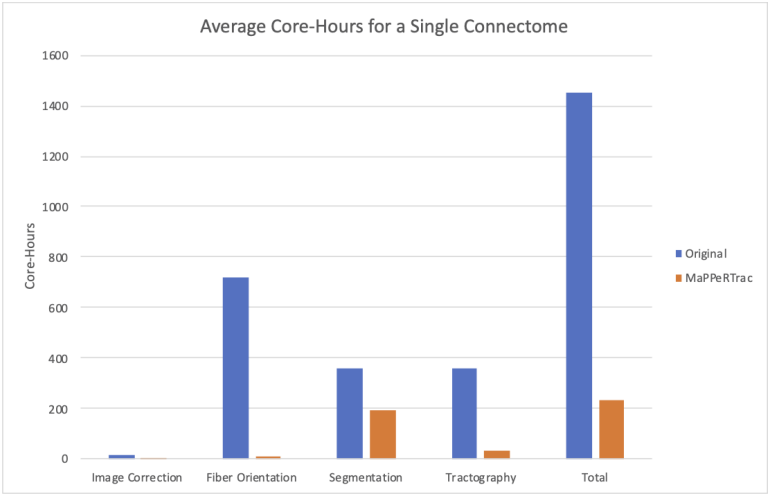

Figure 3: Average resource consumption for a single connectome

6.1. Performance

Compared to the original tractography scripts, we achieve enormous speedups at all stages of tractography. Figure 2 shows improvements to the average time to compute a single connectome by MaPPeRTrac versus

the original scripts, broken down by each step. In the most dramatic case, fiber orientation runs 8600% faster, thanks to GPU acceleration. Image correction and tractography also see significant speedup by spreading sub-tasks across nodes. Segmentation has a more modest speedup, since it can only distribute processes across cores on a single node. The average total speedup for a single patient is 735%, from 40 to 5.5 hours.

For resource-constrained users, a more practical metric is average core-hours. This measures the average compute time for a single connectome multiplied by the number of utilized computational cores. Since much of our performance gains are derived from parallelization, it is necessary to compare how much of this improvement derives from resource availability alone. In particular, tractography consumes a vast amount of core-hours that can only be mitigated by parallelization across many nodes. Nevertheless, MaPPeRTrac on average consumes a total of 625% fewer core-hours.

6.2. Parametrization

We include several examples of configuration files that show the configuration files that enabled parametrized execution of the pipeline on various computational elements in the Appendix A.

6.3. Portability

The pipeline has been run on two supercomputing resources and a university computing cluster with minimal site-specific changes to a single configuration file. The site-specific changes include the location of input files and the bespoke user options that Parsl needs to interact with job schedulers. When executing on a cloud, the configuration options also include account information and location of security credentials. These configuration elements do require a certain technical know-how, which may be barrier to entry.

Nevertheless, our collaborators at multiple institutions have validated the portability and ease-of-use of our pipeline by running it with growing independence.

On all the platforms, the Parsl framework and Singularity container worked flawlessly, validating the portability of the MaPPeRTrac.

6.4. Enabling FAIRness

In line with our objectives, the pipeline embodies and enables the FAIR principles. To ensure the pipeline's artifacts remain Findable and Accessible, we published the source code to GitHub with a permissive BSD license. We also uploaded the container recipe to Singularity Hub, the official registry

for the Singularity software library. Furthermore, the pipeline accepts input identifiers from sources such as OpenNeuro. It generates similar identifiers along with the output artifacts. None of the input, output, or intermediate files are proprietary – the pipeline uses BIDS-compliant open source formats such as NIfTI. We standardized inputs and outputs of the pipeline according to the BIDS format. DICOM sources, NiFTI images, and pipeline derivatives are kept in separate folders, organized according to their subject and session. This makes it easy to compare different timepoints and retests, since they are always located together.As a result, the pipeline can provide a clear provenance to all of its digital outputs.

7. Conclusions and Future Work

Progress in connectomics has been limited by steep computational cost of probabilistic white matter fiber tractography, complexity in installing applications and dependencies, challenges in scaling while adhering to best practices for reproducible research. We created MaPPeRTrac to enable high performance, parallel, parametrized and portable generation of connectomes that is well-tested, robust and easy to use for the community. To lower the barrier to entry for users, we plan to make the pipeline available as a service on public cloud computing resources so any researcher can upload data and generate the connectome using the state-of-the-art tools without becoming a computational expert.

References

[1] T. Behrens, M. Woolrich, M. Jenkinson, H. Johansen-Berg, R. Nunes, S. Clare, P. Matthews, J. Brady, S. Smith, Characterization and propagation of uncertainty in diffusion-weighted mr imaging, Magnetic Resonance in Medicine 50 (2003) 1077–1088. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/mrm.10609. doi:10.1002/mrm.10609. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/mrm.10609.

[2] S. Payabvash, E. M. Palacios, J. P. Owen, M. B. Wang, T. Tavassoli, M. Gerdes, A. Brandes-Aitken, D. Cuneo, E. J. Marco, P. Mukherjee, White matter connectome edge density in children with autism spectrum disorders: Potential imaging biomarkers using machine-learning models, Brain Connectivity 9 (2019) 209–220. URL: https://doi.org/10.1089/brain.2018.0658. doi:10.1089/brain.

2018.0658. arXiv:https://doi.org/10.1089/brain.2018.0658, pMID: 30661372.

[3] J. K. Yue, M. J. Vassar, H. F. Lingsma, S. R. Cooper, D. O. Okonkwo, A. B. Valadka, W. A. Gordon, A. I. R. Maas, P. Mukherjee, E. L. Yuh, A. M. Puccio, D. M. Schnyer, G. T. Manley, T. R. A. C. K.-T. B. I. Investigators, Transforming research and clinical knowledge in traumatic brain injury pilot: multicenter implementation of the common data elements for traumatic brain injury, Journal of neurotrauma 30 (2013) 1831–1844. URL: https://pubmed.ncbi.nlm.nih.gov/23815563. doi:10.1089/neu.2013.2970, 23815563[pmid].

[4] T. M. Madhyastha, N. Koh, T. K. M. Day, M. Hernández-Fernández, A. Kelley, D. J. Peterson, S. Rajan, K. A. Woelfer, J. Wolf, T. J. Grabowski, Running neuroimaging applications on amazon web services: How, when, and at what cost?, Frontiers in neuroinformatics 11 (2017) 63–63. URL: https://pubmed.ncbi.nlm.nih.gov/29163119. doi:10.3389/fninf.2017.00063, 29163119[pmid].

[5] Top500, Top 500 supercomputer sites, https://www.top500.org/, 2020.

[6] T. Roine, B. Jeurissen, D. Perrone, J. Aelterman, W. Philips, J. Sijbers, A. Leemans, Reproducibility and intercorrelation of graph theoretical measures in structural brain connectivity networks, Medical Image Analysis 52 (2019) 56 – 67. URL: http://www.sciencedirect.com/science/article/pii/S1361841518308569. doi:https://doi.org/10.1016/j.media.2018.10.009.

[7] D. E. Rex, J. Q. Ma, A. W. Toga, The loni pipeline processing environment, NeuroImage 19 (2003) 1033 – 1048. URL: http://www.sciencedirect.com/science/article/pii/S105381190300185X. doi:https://doi.org/10.1016/S1053-8119(03)00185-X.

[8] M. Schirner, S. Rothmeier, V. K. Jirsa, A. R. McIntosh, P. Ritter, An automated pipeline for constructing personalized virtual brains from multimodal neuroimaging data, NeuroImage 117 (2015) 343 – 357. URL: http://www.sciencedirect.com/science/article/pii/S1053811915002505. doi:https://doi.org/10.1016/j.neuroimage.2015.03.055.

[9] G. Theaud, J.-C. Houde, A. Boré, F. Rheault, F. Morency, M. Descoteaux, Tractoflow: A robust, efficient and reproducible diffusion mri pipeline leveraging nextflow & singularity, NeuroImage 218 (2020) 116889. URL: http://www.sciencedirect.com/science/article/pii/S105381192030375X. doi:https://doi.org/10.1016/j.neuroimage.2020.116889.

[10] D. Van Essen, K. Ugurbil, E. Auerbach, D. Barch, T. Behrens, R. Bucholz, A. Chang, L. Chen, M. Corbetta, S. Curtiss, S. Della Penna, D. Feinberg, M. Glasser, N. Harel, A. Heath, L. Larson-Prior, D. Marcus, G. Michalareas, S. Moeller, R. Oostenveld, S. Petersen, F. Prior, B. Schlaggar, S. Smith, A. Snyder, J. Xu, E. Yacoub, The human connectome project: A data acquisition perspective, NeuroImage 62 (2012) 2222 – 2231. URL: http://www.sciencedirect.com/science/article/pii/S1053811912001954. doi:https://doi.org/10.1016/j.neuroimage.2012.02.018, connectivity.

[11] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. 't Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M. A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, B. Mons, The fair guiding principles for scientific data management and stewardship, Scientific Data 3 (2016) 160018. URL: https://doi.org/10.1038/sdata.2016.18. doi:10.1038/sdata.2016.18.

[12] J. P. Owen, Y. S. Chang, P. Mukherjee, Edge density imaging: Mapping the anatomic embedding of the structural connectome within the white matter of the human brain, NeuroImage 109 (2015) 402 – 417. URL: http://www.sciencedirect.com/science/article/pii/S1053811915000105. doi:https://doi.org/10.1016/j.neuroimage.2015.01.007.

[13] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. M. Wozniak, I. Foster, M. Wilde,

K. Chard, Parsl: Pervasive parallel programming in python, in: 28th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC), 2019. URL: https://doi.org/10.1145/3307681.3325400. doi:10.1145/3307681.3325400, babuji19parsl.pdf.

[14] M. Hernandez-Fernandez, I. Reguly, S. Jbabdi, M. Giles, S. Smith, S. N. Sotiropoulos, Using gpus to accelerate computational diffusion mri: From microstructure estimation to tractography and connectomes, NeuroImage 188 (2019) 598 – 615. URL: http://www.sciencedirect.com/science/article/pii/S1053811918321591. doi:https://doi.org/10.1016/j.neuroimage.2018.12.015.

Appendix A. Examples for Parametrization

```
    {
    "subjects_json": "example_subjects.json",
    "output_dir": "output",
    "scheduler_name": "slurm",
    "scheduler_bank": "ccp",
    "scheduler_partition": "pbatch",
    "steps": "s1",
    "force": "true"
}
```

```
    {
    "subject_list": "/home/madduri/connectome/2week.txt",
    "output_dir": "/home/madduri/connectome/output/",
    "slurm_bank": "genomics-dl",
    "slurm_partition": "haswell",
    "force": "true",
    "gpu_steps": "s2a s2b",
    "gpu_options": "#SBATCH --gres=gpu:1",
    "s2a_nodes": "3",
    "s2a_cores": "16",
    "s2b_nodes": "3",
    "s2b_cores": "16",
    "s3_nodes": "4",
    "s3_cores": "16",
    "s3_nodes": "4",
    "s3_cores": "16",
```

```
    "s4_nodes": "4",
    "s4_cores": "16",
    "steps": "s1 s2a s2b s3 s4 s5"
    }

    {
    "subject_list": "/home/madduri/connectome/GO.txt",
    "output_dir": "/blues/gpfs/globalscratch/madduri/GO/output/",
    "slurm_bank": "genomics-dl",
    "slurm_partition": "biggpu",
    "force": "true",
    "gpu_steps": "s2a",
    "gpu_options": "#SBATCH --gres=gpu:2",
    "s2a_nodes": "1",
    "s2a_cores": "16",
    "s2b_nodes": "1",
    "s2b_cores": "16",
    "s3_nodes": "4",
    "s3_cores": "16",
    "s3_nodes": "4",
    "s3_cores": "16",
    "s4_nodes": "4",
    "s4_cores": "16",
    "steps": "s1 s2a s2b s3 s4 s5"
}

{
    "subjects_json": "subjects_new.json",
    "output_dir": "output",
    "scheduler_name": "aws",
    "scheduler_bank": "ccp",
    "scheduler_partition": "pbatch",
    "steps": "s2a",
    "gpu_steps":"s2a",
    "container_path":"/home/ubuntu/MaPPeRTrac/container/image_old.simg",
    "force": "true",
    "worker_init": "apt-get update; apt-get install -y nfs-common;
    mkdir -p /home/ubuntu/MaPPeRTrac; echo '192.168.167.0:/home/ubuntu/MaPPeRTrac
    /home/ubuntu/MaPPeRTrac nfs4 auto 0 0' >> /etc/fstab;
    mkdir -p /home/ubuntu/subjects; echo '192.168.167.0:
```

```
    /home/ubuntu/subjects /home/ubuntu/subjects nfs4 auto 0 0'
    >> /etc/fstab;mount -a;
    echo '192.168.167.0 ip-192-168-167-0.ec2.internal headnode' >> /etc/hosts"
}
```

Appendix B. Availability of the pipeline and container

The pipeline and instructions to the run are available from GitHub (https://github.com/LLNL/MaPPeRTrac). The pipeline divided into steps, which can be run individually or sequentially

1. s1_dti_preproc.py – correct motion, eddy current, noise: Explanation DTI process- ing:BET: brain Extraction Tool (Smith, 2002), Eddy_correction: Images corrected for mo- tion and eddy currents using the FMRIB linear-image registration tool (FLIRT) with a 12-parameter linear image registration (Jenkinsonet al., 2002) using the b = 0 s/mm2 image as the reference. DTIfit: FA image calculation. Using FLIRT, the FA map of every subject and session was registered to the T1 in order to obtain a diffusion to structural transform.
2. s2a_bedpostx.py - The Edge Density is then computed as a simple count of the number of edges that pass through each voxel, by adding together the binarized tractography results.stx for fiber estimation–setup diffusion parameters - Estimates the fiber orientation at every voxel (with a maximum number of two and other default settings
3. s2b_freesurfer.py - for cortical parcellation – T1 Segmentation using the Desikan–Killiany atlas (Desikan et al., 2006) from FreeSurfer resulting in 68 cortical regions and 14 subcortical regions. The 68 cortical regions are transformed to the gray–white matter boundary (GWB). These 82 regions represent the nodes of the connectome. Additionally, we register the FA volume to the T1 volume. Each of the cortical GWB volumes and the subcortical volumes was registered to the diffusion space to be used as seeds for the tractography. Both s2* steps can be run in parallel
4. s3_probtrackx.py – probabilistic diffusion, generates connectomes and EDI. We generate 200 streamlines from each seed voxel, with the target volume used as a waypoint mask and all 80 other volumes excluded. 82*81 = 6642 tractography runs per subject. Tractography results are binarized to create a mask of white matter voxels needed to connect each pair of cortical/subcortical regions. This uses a "consensus connectome" based on [2]. The resulting connectome can be processed

into an edge density image (EDI), which counts of the number of edges that pass through each voxel.

Appendix C. Additional computational experiments

1. NVRAM disk to accelerate file IO - we ran the pipeline using NVRAM memory on the Catalyst HPC cluster. This resulted in negligible performance improvement.
2. Simplify tractography - rather than run tractography between every single pair of regions, we attempted to run tractography a single time for each region and calculate connectivity post-hoc. However, this failed to capture sufficient edges and proved to be poorly supported by the Probtrackx software.
3. Dynamic job times - we tried to dynamically configure job times to help gain priority on computing queues. This proved unwieldy in practice.
4. Freesurfer GPU acceleration - the slowest remaining part of the pipeline is Freesurfer segmentation. We used pre-compiled binaries developed by [14] to use GPU acceleration for Freesurfer. However, this crashed on all Linux systems we tested in more than 90% of cases. Even when it ran successfully, speedup was only a modest 10-20%.
5. Automatic visualization - we wrote an additional pipeline step, s4_render.py, to render edge density images generated by the pipeline. Image rendering is even possible on headless nodes as we included the VTK library with the container.